

Versatile virtual honeynet management framework

ISSN 1751-8709

Received on 24th June 2015

Revised 28th December 2015

Accepted on 4th February 2016

doi: 10.1049/iet-ifs.2015.0256

www.ietdl.org

Wenjun Fan¹ ✉, David Fernández¹, Zhihui Du²¹Departamento de Ingeniería de Sistemas Telemáticos, ETSI Telecomunicación, Universidad Politécnica de Madrid, 28040 Madrid, Spain²Department of Computer Science and Technology, Tsinghua National Laboratory for Information Science and Technology, Tsinghua University, Beijing 100084, People's Republic of China

✉ E-mail: efan@dit.upm.es

Abstract: Honeypots are designed to investigate malicious behaviour. Each type of homogeneous honeypot system has its own characteristics in respect of specific security functionality, and also suffers functional drawbacks that restrict its application scenario. In practical scenarios, therefore, security researchers always need to apply heterogeneous honeypots to cope with different attacks. However, there is a lack of general tools or platforms that can support versatile honeynet deployment in order to investigate the malicious behavior. In this study, the authors propose a versatile virtual honeynet management tool to address this problem. It is a flexible tool that offers security researchers the versatility to deploy various types of honeypots. It can also generate and manage the virtual honeynet through a dynamic configuration approach adapting to the mutable network environment. The experimental results demonstrate that this tool is effective to perform automated honeynet deployment toward a variety of heterogeneous honeypots.

1 Introduction

A honeypot is an information system resource whose value lies in unauthorised or illicit use of that resource [1]. The honeypots are often employed to investigate malicious behaviour. However, every individual honeypot has its own features.

First, some honeypots such as Honeyd [2] can emulate multiple decoys simultaneously to monitor the unauthorised traffic. These decoys can emulate the appearance of operating systems and vulnerable services, but they provide little interaction to the adversaries. This kind of honeypot is called low-interaction honeypot (LIH). Second, medium-interaction honeypot (MIH), e.g. Amun [3] and Dionaea ['Dionaea – caught bugs', <http://www.dionaea.carnivore.it/>], can provide much more interaction to the adversaries and even catch the malicious payload. They can emulate a variety of vulnerable services based on the TCP/IP network stacks which are implemented and managed by the underlying operating system where the MIH installs. However, it is limited by the fact that it emulates known vulnerabilities, and its security program only focuses on capturing the malicious traffic accessing to its emulated vulnerable services. Third, a genuine computer system running as a honeypot is called high-interaction honeypot (HIH), since it can provide a fully functional operating system to the adversaries. Using HIHs, security researchers can capture not only the network activity, but also the system activity. The limitation of HIHs is the resource consumption for large-scale deployment. Therefore, every type of honeypot has its own value for specific security goal, however, as well as has its own functional limitation. As such, the probable approach to solve the limitation mentioned above is to build a platform that is flexible enough to support versatile honeynet deployment in order to trap and investigate the attacks with appropriate decoys.

Thus, there are at least two technological considerations regarding honeynet deployment. First, compared with deploying physical honeynet which is not used widely since its management limitation and resource cost, using virtualised tools is more efficient in term of resource usage. Second, the production network environment is mutable, thus the honeynet deployment should be able to configure dynamically to adapt to the real time production network environment.

However, it is a tedious task to configure and generate a virtual honeynet manually, and even a complex task if the virtual honeynet

is required to adapt to the network environment instantly. So, it is necessary to an automated tool to configure, deploy and manage a flexible honeynet. Therefore, we propose a versatile virtual honeynet enabling tool, Honeyvers, to address these problems. The main contribution of Honeyvers is to provide a coherent implementation that meets these three requirements:

- It can manage the deployment of heterogeneous honeypots.
- It can facilitate the dynamic deployment for both LIHs and HIHs.
- It can deploy honeynets for different security goals.

The organisation of this paper is as follows: in Section 2, some related work is discussed; in Section 3, an overview of Honeyvers architecture is presented; in Section 4, the Technology Independent Honeynet Description Language (TIHDL) is briefly described; in Section 5, the transformation methodology is described in detail; in Section 6, the dynamic deployment tools are introduced; in Section 7, one kind of sensor is devised; in Section 8, honeynet deployment is discussed; in Section 9, several experimental results are demonstrated; and in Section 10, some conclusions are presented.

2 Related work

Dynamic honeynet can greatly improve the deployment and maintenance of decoys by monitoring and learning the organisation networks in real time. Though it is a challenge to achieve dynamic honeynet configuration, there are several proposals that had facilitated this idea [4, 5].

The most recent proposal about dynamic honeynet is a new automated honeynet deployment for dynamic network environment [6]. The authors advocated using honeypot as a representative system to collect interesting traffic data for forensics analysis. Thus, they developed a honeynet that is capable of dynamically configuring a representative honeynet involving the combination of both passive and active scanning. They considered the scanning noise as the factor of the representative accuracy. The experimental results demonstrated that the different scanning noise levels lead to different time-consuming and representative accuracy. This proposal focuses on the deployment strategies of dynamic honeynet by applying the strategies to LIH, i.e. Honeyd, but did not consider the HIH dynamic deployment.

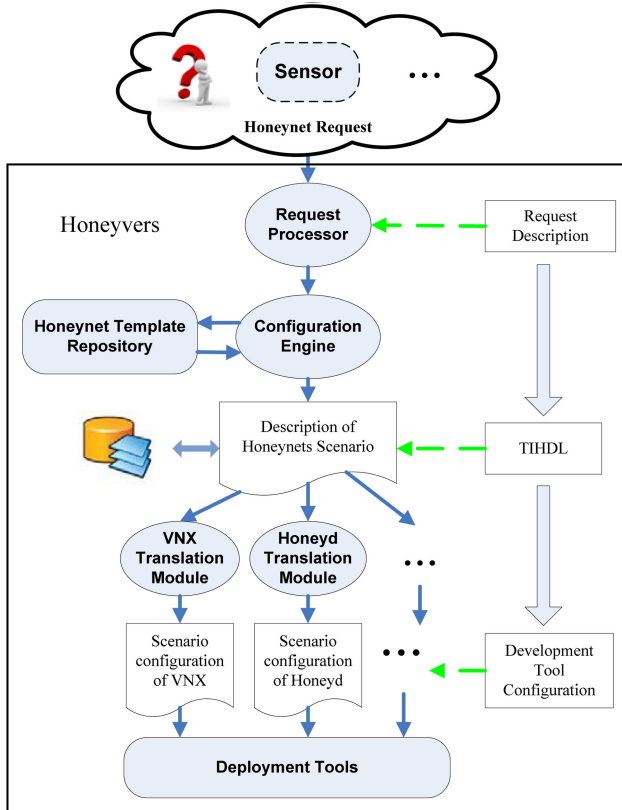


Fig. 1 Overview of Honeyvers architecture

The proposals mentioned above focus on dynamic deployment of LIHs. However, LIH has fidelity limitation. HIHs can guarantee fidelity, but it is difficult to deploy them on large scale due to the resource and management restraint. A hybrid system: namely, Potemkin [7] was devised as a way using generic routing encapsulation (GRE) tunnel to deploy scalable and dynamic HIHs. It employs a network gateway, to which routers all over the Internet are configured to tunnel an address prefix, as an agent to take responsibility of sending traffic to a honeyfarm server. The gateway instructs virtual machine monitor (VMM) that runs on each physical server to create a new HIH on demand for each active destination IP address. Otherwise, if one HIH is idle, the VMM will destroy it and reclaim the resources when being instructed by the gateway. However, it is not an open-source project hence we cannot gain insight into the implementation.

Nevertheless, currently there are several tools that can be used to deploy dynamic honeynets. Smartfrog [8] is a powerful and flexible Java-based framework for configuring, deploying and managing distributed software systems. It has multiple software components across a network of computing resources. Smartfrog proposes a simple text-based name-value language for scenario dynamic configuration, which enables the automatically install, start and shut down of the systems. Though the Smartfrog language is not generic, it can facilitate dynamic honeynet deployment.

Most recently, some new virtualisation technologies were proposed to facilitate more capable honeynet architectures. For instance, the decoys based on kernel-based virtual machine (KVM) hypervisor [9] can provide low-level surveillance from outside the guest OS [10], which can keep a stealthy monitoring to the system activity and the intruder has no way to bypass that surveillance. Besides, Linux containers (LXC) virtualisation provides a lightweight alternative to hypervisor-based virtualisation [11]. LXC can create multiple isolated Linux user-space instances by partitioning the resource of the host. Thus, the startup of an LXC-based virtual machine is much quicker than that based on KVM, but LXC can only emulate Linux over Linux, but not other operating systems.

3 Overview of Honeyvers architecture

In this section, Honeyvers standing for versatile honeynet is presented. Honeyvers consists of six components (see Fig. 1). These components should work together to accomplish a complete honeynet deployment.

Through the whole Honeyvers architecture, the six major components are the sensor, the request processor, the configuration engine, the honeynet template repository, the specific translation modules and the deployment tools. Besides, the request description, TIHDL [12] and the development tool configuration are employed separately at each step for the honeynet generation.

The request for honeynet can be made manually by security researchers or automatically by the sensor. The sensor is not a compulsory component of our tool, but is a supplementary client side of Honeyvers. As soon as the tool receives the input, the request processor first will check the request syntax. If the request syntax is correct, then the configuration engine will process the request content. If the request calls for a template, the configuration engine will apply the requested template in the repository. Otherwise, the configuration engine will make a complete honeynet description according to the content-input. Afterwards, the corresponding translation module will interpret the general honeynet description into specific configuration for the target deployment tool. At last, the deployment tools deploy the requested honeynet by processing the corresponding specific configuration.

The functions of the Honeyvers lead to some research and technological challenges which are summarised by the following questions:

- How to support versatile honeypots with a general tool?
- How to facilitate the dynamic deployment?
- What is the honeynet deployment scale depending on different platforms?

These three questions are answered in the following sections.

4 Technology independent honeynet description language

TIHDL is a general language designed to describe the honeynets, covering the specific characteristics in terms of service, operating system and network topology. It is defined being independent of the platform where the honeynet will be automatically deployed. TIHDL inherits and evolves from the data model of the common information model, which is technology independent and can provide descriptions for almost every aspect of the computer networks. TIHDL also follows the generic description style of Honeyd which proposed a virtual honeynet BNF-syntax-based description language providing general descriptions for honeynet. Furthermore, TIHDL emulates the schema of the configuration language of VNX [13], which provides an extensible markup language (XML) syntax-based virtual computer network configuration language.

5 Transformation methodology

The versatility of Honeyvers is achieved through two major capabilities: first, it allows the honeynet scenario to be described by TIHDL; second, it accepts various existing deployment platforms to be plugged in. This section proposes a methodology for transforming TIHDL into other platform configurations. The TIHDL-based general configuration file must be transformed to a specific configuration file which can be further recognised by specific deployment tool. The transformation methodology is graphically presented in Fig. 2.

Since TIHDL uses XML-syntax, we apply the schema mapping methodology for data integration and transformation. The transformation methodology comprises of four steps. The task at each step is accomplished by specific tools, our methodology, however, is not limited to these proposed tools:

Schema mapping and transformation generator: These two tasks can be performed simultaneously by using the comprehensive

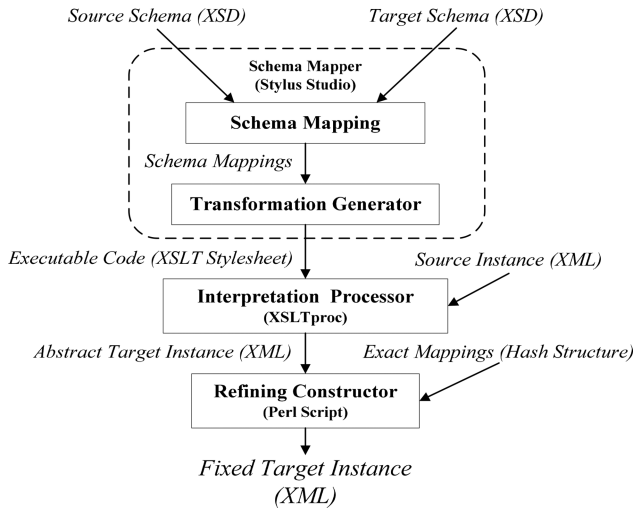


Fig. 2 Transformation methodology

```

XPath_TargetElement1 {
  GeneralValue1 => SpecificValue1;
  GeneralValue2 => SpecificValue2;
}
XPath_TargetElement2 {
  GeneralValue1 => SpecificValue1;
  GeneralValue2 => SpecificValue2;
}
  
```

Fig. 3 Hash structure

Algorithm

1. GET the hash structure
2. GET the abstract target instance
3. FOR each element of the abstract target instance
4. IF element = one block name of the hash structure
5. FOR each key in the block
6. IF element equal to one key
7. SET element value to the key value
8. ENDIF
9. ENDFOR
10. ENDIF
11. ENDFOR
12. DISPLAY the fixed target instance

Fig. 4 Algorithm of refining constructor

XML toolkit called Stylus Studio [http://www.stylusstudio.com/]. We apply its schema mapper to the source schema and the target schema. As we know, schema mapping is a complex task. The schema mapping performed between two schemas with the same level of granularity is called horizontal schema mapping. The schema mapping otherwise from a general schema into a specific schema is named vertical schema mapping. Currently, there are many proposals about automated schema mapping. Considering that the automated schema mapping works well for horizontal mapping, but is incapable of vertical mapping, we employ Stylus Studio which can facilitate a variety of schema mappings despite its deficiency in providing automated mapping. After the schema mapping, the transformation generator will create the executable code in XSLT-stylesheet.

Interpretation processor: At this step, the input, i.e. the XML-based source instance is interpreted according to the transformation executable code. We employ the XSLTproc [http://www.xmlsoft.org/XSLT/xsltproc.html] as the interpretation processor. The output of this step is an abstract target instance because the exact target instance needs the specific technology dependent data; however, at this step only the general data from the source instance are transformed and integrated.

Refining constructor: This is the last step aimed to generate the exact target instance which can be recognised by the deployment tool. For each specific deployment tool, we must prepare an exact mapping in order to transform the general data into specific data. The refining constructor is a programmable component which currently uses Perl scripts to refine the abstract target instance by extracting the specific data from the exact mapping hash structure. The hash structure is as follows: (see Fig. 3).

In the hash structure, every block is named by the XPath of the target element whose general value needs to be concretised. In each block, the key-value pair is composed of the general value and the specific value. The refining algorithm is as follows: (see Fig. 4).

We apply this transformation methodology to our translation modules. Each translation module consists of the interpretation processor, the refining constructor, the executable code and the exact mappings. The executable code is generated by schema mapper, which is not a simple task. However once it is made, it can be used permanently.

6 Dynamic deployment tools

Honeyd and VNX are employed by Honeyvers as the virtualised tools to take the responsibility of dynamic deploying heterogeneous honeypots.

Honeyd can emulate multiple decoys simultaneously following a certain network topology. It has a doorway called Honeydctl to communicate its inner workings. Honeydctl can be used to reconfigure the templates dynamically. All commands used in the regular configuration file of Honeyd template can be interactively used after 'honeydctl>' prompt; however, this is also the weakness of Honeydctl: the user has to input the commands interactively, one by one. Instead of interacting with Honeydctl, Honeyd allows users to reconfigure the Honeyd templates through a UNIX socket located in /var/run/honeyd.sock. To apply this function, we developed a client socket used to send scripts consisting of Honeyd commands.

On the other hand, there are a lot of MIH software can emulate vulnerable services, e.g. Dionaea and Amun, which need a deployment tool to hold them. Moreover, a virtual HIH also needs guest virtual machine as the deployment carrier to contain it. Thus, VNX is employed since (i) its scalability in creating virtual machines due to the ability to deploy virtual network scenarios over a cluster of servers; (ii) its ability to individually reconfigure any decoy dynamically without influencing other decoys; and (iii) it integrates multiple hypervisors such as KVM and LXC and can also supply dynamic configuration.

7 Design of sensor

Our system can accept manual input/request, but handwriting is a tedious task. Automated request creation is required in many scenarios. Sensor is a component which is used to generate automated input by scanning the target production network. The scan process is shown in Fig. 5.

In that we focus on deploying honeypots assigned with free IP addresses. The objective of this sensor is to generate the corresponding honeynet description through scanning a target production network.

The scan process starts with probing engine issuing a port/OS to a target production network. Our tool is developed based on Nmap [http://www.nmap.org/]; thus, we employ the Perl module: namely, Nmap::Parser as the core of probing engine to facilitate this task. The probing engine can probe the target production network (e.g. nmap -sT -PR -O 192.168.1.*), and collect data including IP address, operating system, services and open ports. Thanks to Nmap::Parser, the data of production systems can be generated as XML output. Afterwards, the sensor uses TIHDL to formalise the data and generates a complete honeynet description by assigning free IP addresses to the honeypots.

Subsequently, the sensor compares the current honeynet description to the stored one. If the current scan is the initial one and there is nothing stored previously, then the current description is stored and the initial request is produced. Otherwise, the current

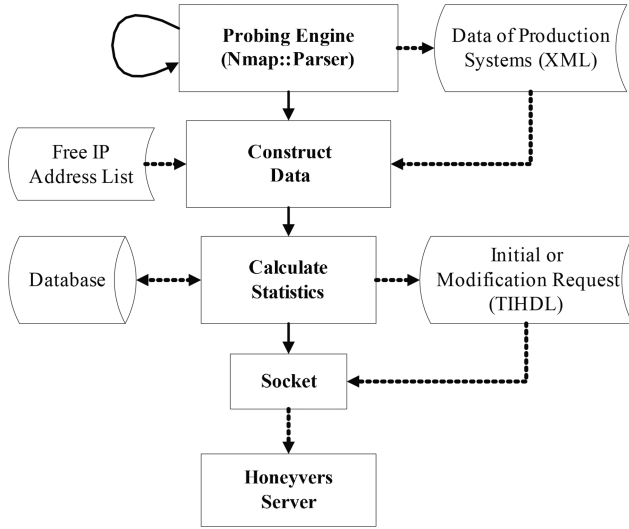


Fig. 5 Scan process. The thick lines correspond to control channel events while the dashed lines denote flow of data

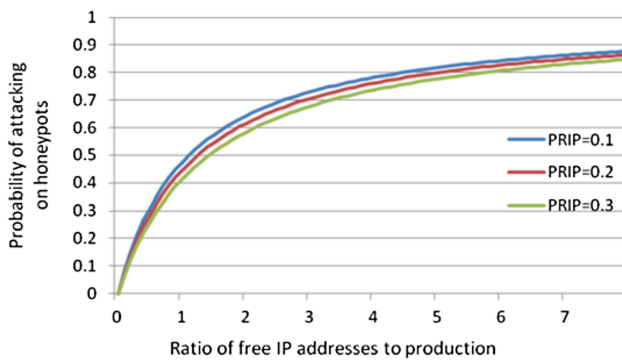


Fig. 6 Probability of attacking on honeypots in the network versus the ratio of free IP addresses to the NPS

scan will be compared with the stored one. If the comparison result shows that the current description is different from the stored one, the sensor will calculate the statistics of the elements such as the honeypots, the services and the ports, which should be added or removed, and then produce a modification request with TIHDL, and at the same time the sensor will update the database.

Consequently, if there is a request generated, the sensor will send it to Honeyvers through a socket. Meanwhile, the sensor scans the target network repeatedly at intervals.

This Nmap-based sensor is just one case of application, it has limitation in getting a complete topology from scanning a network which has more than one hop, but it can work well for scanning the flat network topology for deploying honeynet integrated into the production network with the unused IP addresses.

8 Honeynet deployment for diverse platforms

In this section, we discuss the honeynet deployment for different deployment platforms. The honeynet deployment depends on two conditions: the security goal of the honeypot and the performance of the platform.

8.1 Deployment of Honeyd

Honeyd is always used to monitor the unused IP addresses in order to capture interesting traffic as many as possible, since the network traffic coming to the unused IP addresses seem to be suspicious naturally. The honeypots assigned with unused IP addresses can deceive attackers to spend time and resource attacking decoys rather than production systems, thus they can effectively deter attackers. Therefore, the probability of attacking on the honeypots is increasing with the ratio of honeypots to production systems.

A Honeyd template can emulate both the operating system and the services. Thus, every production system must use the operating system, the service and the port information for identification. Specifically, we assume that NPS denotes the total number of production systems, NPS_i represents the i th production system, $NFIP$ means the number of free IP addresses and P_{RIP} is the percentage of free IP addresses which should be kept for future use. Thus, the number of virtual honeypots (NVHs) used to emulate the i th production system is

$$NVH_{PS_i} = \left(\frac{NPS_i}{NPS} \right) \times NFIP \times (1 - P_{RIP}), \quad (1)$$

$$\text{s.t. } NFIP \times (1 - P_{RIP}) \geq NPS$$

We restrict that $NFIP \times (1 - P_{RIP}) \geq NPS$ so that there would be enough IP addresses to emulate production systems. When honeypots are integrated into the production networks, the possibility of attacks on honeypots is

$$P_{VH} = \sum_i NVH_{PS_i} / (NPS + \sum_i NVH_{PS_i}) \quad (2)$$

We note that $NPS = \sum_i NPS_i$, thus we can substitute the value of NVH_{PS_i} in (1) for that in (2). We can obtain

$$P_{VH} = NFIP(1 - P_{RIP}) / (NPS + NFIP(1 - P_{RIP})) \quad (3)$$

Fig. 6 plots P_{VH} versus the ratio of free IP addresses to production systems.

The curve shows the relationship is exponential with the NFIP addresses. If the NFIP addresses are the same as that of production systems, the probability of attacking on production systems is at least 50%. Note that this probability is mainly focusing on automated attacks and script kiddies using automated attacking tools, because a serious advanced attacker will do reconnaissance and quickly detect the LIHs and also the true production systems.

8.2 Deployment of LXC-based honeypots

The honeypot used as deception to attackers fails against automated attacks, which can attack both the honeypots and the production systems of an organisation in a short time. Automated malware focuses on service vulnerabilities as targets of opportunity without regarding for OS fingerprinting. Correspondingly, MIH can emulate multiple well-known vulnerable services in order to catch automated malware and make some fake response on application layer.

LXC stands for Linux containers, which shares the host kernel to operate the virtual machines. The LXC-based virtual machine can facilitate the deployment for those MIH software in order to catch automated malware.

The number of vulnerable services emulated by the honeypot installed on an LXC file system is limited. The honeypot such as Dionaea or Amun only emulates a number of common services. We assume that the LXC-based honeypots can catch the attacks from other infected production systems. Thus, if a honeypot wants to capture more malware, it has to improve the coverage of vulnerable services of the production networks. Therefore, the coverage of vulnerable services: namely, CG_{VS} can be calculated by (4)

$$CG_{VS} = \frac{NSERV_{EMU}}{NSERV_{TOTAL}} \quad (4)$$

$NSERV_{EMU}$ denotes the number of vulnerable service types of a production network, which the honeypots can emulate, and $NSERV_{TOTAL}$ represents the total number of vulnerable service types of the production network.

On the other hand, the number of honeypots depends on the load of concurrent attacks as shown by (5)

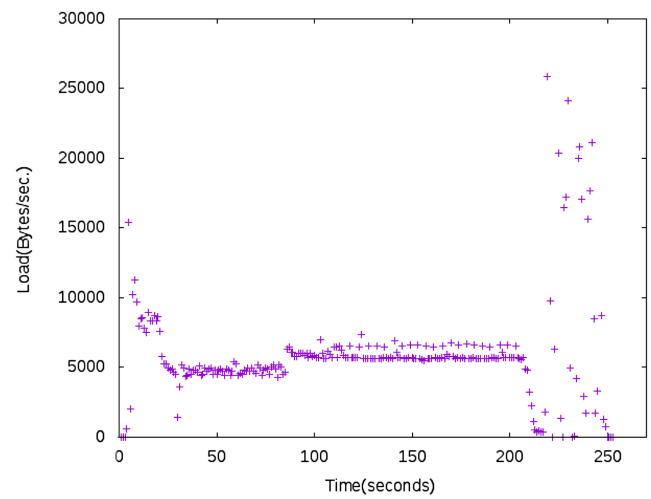
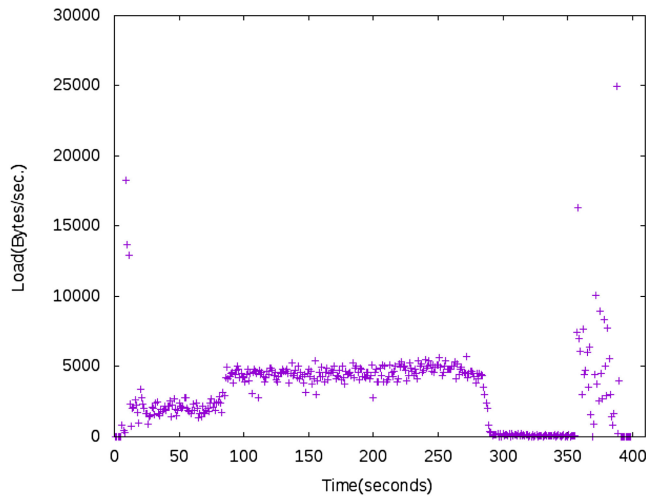


Fig. 7 Port scans of normal mode

$$NVH \times PF_{VH} \geq LD_{CA} \quad (5)$$

NVH means the number of virtual honeypots, PF_{VH} denotes the performance of sustaining concurrent attacks by an LXC-based virtual honeypot, and LD_{CA} is the total count of concurrent attack load. We note that the value of PF_{VH} will decrease if NVH increases, but the relationship between them is not a simple inverse ratio. We present their inverse relationship in Section 9.

8.3 Deployment of KVM-based honeypots

Not all of the threats come from automated attacks. The human-generated attacks always focus exclusively on several specific victims. Before attacking the victim, an adversary must make a reconnaissance. However, making a large-scale fingerprint scan is a tedious work and it is currently infeasible to scan an IPv6 network. Actually, except for scanning the target system, there are several other methods for reconnaissance. For example, the adversary can get the victim's information through DNS server.

In the case of human-generated attack, the target victim is elaborately selected by the attacker. The attacker may alter the system logs or even launch new attacks to other production systems through this compromised one. Thus, it is necessary to monitor the system activities generated by attackers. The HIHs are able to accomplish this task. However, it is not necessary to deploy so many HIHs assigned with unused IP addresses, because the heavily occupied IP addresses of a network will also cause the attacker's suspicion.

Importantly, we should guarantee the fidelity of the HIH. There are two ways to improve the fidelity. First, it is necessary to be able to emulate as many different types of operating systems as possible. KVM hypervisor can emulate any guest OS under x86 architecture. Thus, KVM can guarantee the versatile emulation. Second, as HIHs are used to capture the system activities, we should guarantee the performance of each KVM-based virtual honeypot. However, the number of KVM-based honeypots is

Table 1 Operating system distribution

Linux 3.7–3.9	Linux 2.6.13–2.6.38	Windows Server 2008 Beta 3	Windows Vista or Win7	VxWorks 6.5	Android 4.1.1	Other unknown OS
5%	23%	5%	5%	9%	10%	53%

Table 2 Most popular open ports of the entire production network

Port number	22	80	111	443	445	631	902	5666	7200	8080
counting	14	17	5	5	2	2	3	2	5	2

inversely correlated with the performance of each honeypot. This relationship is demonstrated in Section 9.

9 Case study

9.1 Testbed

The system parameters of the host node are: central processing unit (CPU), 4 Intel(R) Core(TM) i5-3470 CPU at 3.20 GHz; random access memory (RAM), 16 GB; OS, Ubuntu 14.04 LTS; Kernel, Linux 3.13.0-24-generic. This Honeyvers server is integrated into a production network. The network includes computers belonging to faculty, staff and students. After an initial scan on the network, the information about the IP addresses and operating systems is produced: 16% of IP addresses are occupied by production systems and 84% of them are unused. It indicates that numerous virtual honeypots can be mined into the production network. In that the ratio of unused IP addresses to production systems is close to 5, which means the probability of attacking on honeypots is close to 80% according to Fig. 6.

Table 1 illustrates the distribution of operating systems available in the production network. Accordingly, the decoys emulated by Honeyd could be assigned with these operating systems in proportion to what is shown in this table. Besides, at least two types of KVM-based file system (Windows and Linux) should be prepared to set up HIHs.

Table 2 summarises the open ports that appear more than two times among the production systems. The most popular ports are bound up with services http and ssh. If the LXC-based honeypot can emulate all of these open ports, the decoy will get a high probability of catching malware.

9.2 Nmap load test

In that Nmap is an active fingerprint tool, it will generate traffic load into the production network. Nmap offers six levels of timing templates which influence the timing and performance. We test the Nmap load using the two most common modes: normal and polite. The TCP SYN scan is used to perform the port scan. The command is such as this: `Nmap -sS -P0 -O 138.4.7.128/25`. The load tests using normal mode are demonstrated in Fig. 7.

The left part of Fig. 7 shows a natural normal SYN scan, which allows the Nmap itself to adjust the scan timing and performance. This approach generates a network load of ~5 kB/s which lasts about 200 s. However, the complete scan takes 385.25 s.

Subsequently, we tested the normal SYN scan with 20 parallel scans. The result is shown on the right part of Fig. 7. The scan period is 245.19 s, which is shorter than the previous one. However, this approach generates much more network load, which even roughly remains at 10 kB/s at the beginning of the scan. After 25 s, the load reaches around 5 kB/s, and at the interval between 90 and 210 s, the load still keeps at more than 6 kB/s.

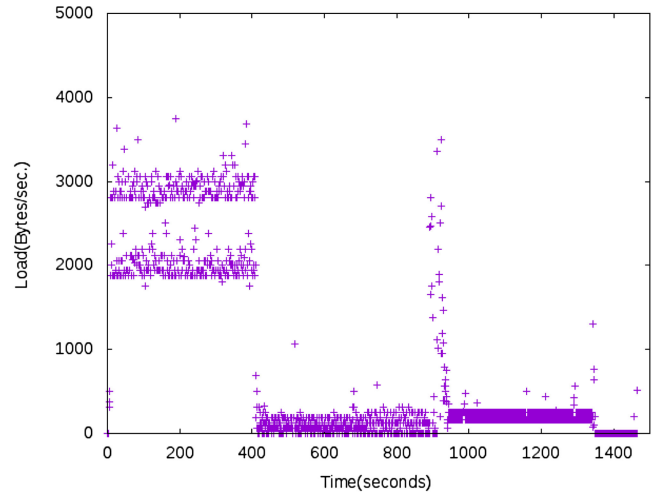
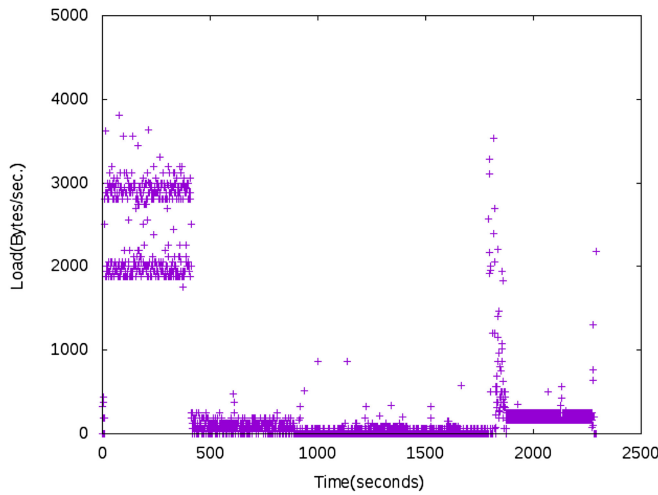


Fig. 8 Port scans of polite mode

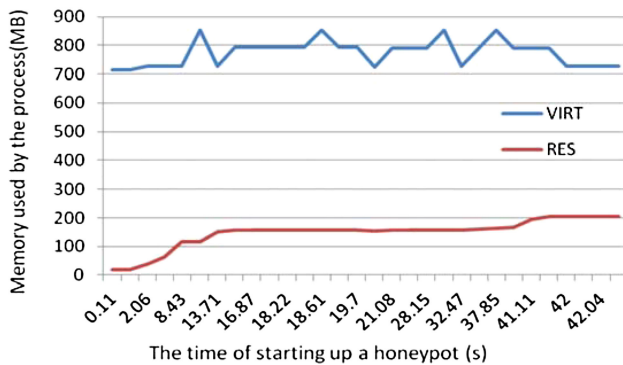


Fig. 9 Virtual and physical memory used by the process

Different from the normal mode, the polite mode serialises the individual port scans for each IP with a delay of 400 ms between scans. The objective of this approach is to decrease the overhead imposed on the network. However, it costs a long time to finish a complete scan. The left part of Fig. 8 shows the scan result via the polite mode with 20 parallel scans.

So, the left part shows that the highest load is about 3 kB/s. After about 500 s, the load decrease to <0.4 kB/s. The complete scan takes 2277.04 s. To shorten the scan period, we increased the number of parallel scans to 40, and the result is shown on the right part of Fig. 8. Then, the complete scan only takes 1344.91 s, which is less than that of using 20 parallel scans.

Therefore, these two modes have their own advantages and disadvantages. We recommend that if the production network is a server network, which is relatively stable, but has heavy network load, it would be better to use polite mode to scan the server network. However, if the production network includes lots of client systems, which can evoke mutable network environment, then we could apply the normal mode to scan the production network as soon as possible.

9.3 LXC performance test

VNX can deploy LXC-based honeypots which can only emulate the Linux operation system. Thus, this method lacks fidelity.

Table 3 Apache benchmark results based on LXC

Number of honeypots	5	10	15	20
Requests per second	19,309.32	18,395.51	18,008.72	17,832.99

Table 4 Host performance with ten LXC-based honeypots

TIME+	RES	VIRT	%CPU	%MEM
0:00.68	37M	168M	22.5	0.2

However, the LXC-based virtual machine is very suitable for the MIHs such as Dionaea and Amun.

Security is tightly coupled with performance, thus we use the Phoronix Test Suite (PTS) [http://www.phoronix-test-suite.com/] to test the performance of virtual honeypots. PTS is an open-source benchmark platform which has been used to benchmark and compare various system attributes.

Table 2 shows that the most common port 80 binds to http service. Thus, we use Apache benchmark of PTS to test the LXC-based honeypots. Apache benchmark measures how many requests a given system can sustain per second when carrying out 500,000 requests with 100 being carried out concurrently. This benchmark is chosen as it provides insight on how the LXC-based honeypot is able to handle increasing input/output stress in terms of CPU and memory usage when the NVHs increases.

Table 3 shows how the increasing number of virtual machines affects the performance. Our Honeyvers server itself can sustain 20,656.09 requests per second. In Table 3, the performance declines as the number of concurrent honeypots increases. However, even though the host runs 20 virtual honeypots, it still has a good performance in terms of request sustain.

We also tested the host performance of deploying virtual honeypots. We used the Linux system command 'top' to monitor the system performance when deploying ten LXC-based honeypots. The test results are shown in Table 4.

The results show that the startup of LCX-based virtual honeypots is very fast, <1 s for ten VMs to boot up, and the resource occupation is also quite low. So, if fidelity is not the most important concern, the LXC-based MIH is a better choice.

9.4 KVM performance test

VNX can also deploy KVM-based HIH which can emulate various operating systems. The problem of this method is also the performance. We should guarantee the performance of each KVM-based honeypot to improve the fidelity. However, the number of KVM-based virtual machines that can be sustained by the host is limited due to the host resource. Thus, we test both the performance of the host and the virtual machine.

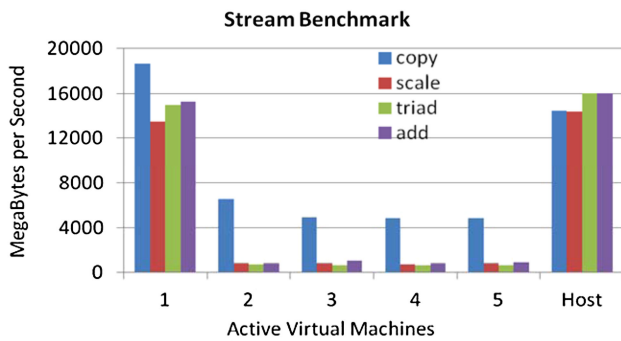
The parameters of the virtual machines are: CPU, 1 Intel(R) Core(TM)2 Duo CPU T7700 at 2.40 GHz; RAM, 384 MB; OS, Ubuntu 14.04 LTS; Kernel, Linux 3.13.0-24-generic.

First, we observed the system performance when launching only one honeypot on the host machine. It costs 40.02 s to startup one honeypot. The memory used by the process during this experiment is demonstrated in Fig. 9.

The virtual memory usage is not stable. After the honeypot is switched on, the usage of virtual memory decreases to 726 MB. However, the physical memory usage is quite steady after 16.87 s. It maintains at the level of around 160 MB, and in the end it stays at 206 MB.

Table 5 Host performance with ten KVM-based honeypots

TIME+	RES	VIRT	%CPU	%MEM
1:08	206M	864M	118	1.3

**Fig. 10** Stream benchmark results based on KVM**Table 6** Statistic of data capture

	Incidents/hits	Successfully attacked port (counts)	The two busiest attackers (counts)
Dionaea	6758/21	21(14), 5060(5), 42(1), 32,847(1)	83.34.220.39 (4), 61.216.2.14 (3)
Amun	278/15	21(15)	61.216.2.14 (4), 62.210.207.107 (4)

Second, we deployed 10 KVM-based honeypots. The host launched ten processes, and each process is corresponded to one honeypot. The largest values of the five parameters of command 'top' are resulted from these ten processes in Table 5.

The largest parameter values of these ten processes are more or less the same as those we got when launching one. Though VNX sequentially launches the KVM-based machines, the total startup time for ten HIHs is not a simple accumulation of the total time of each active process. The total startup time for ten KVM-based honeypots is around 5 min.

On the other hand, we consider the performance of the KVM-based HIHs, because many adversaries can detect the honeypots by testing the system performance. In that it is probable for the adversary to read and write the system resource, we use the Stream benchmark of PTS to test performance of the KVM-based honeypots. The Stream benchmark of PTS is built from a programme which is designed to measure the sustainable memory bandwidth rather than the burst or peak performance of a machine. The Stream benchmark has four operating modes: copy, scale, triad and add. Fig. 10 shows how the increasing number of honeypots affects the performance of the four operating modes.

With the NVHs increasing, only the copy mode still performs in an acceptable range, the other three modes however show very poor performance. The reason is that the other three modes rely more heavily on the CPU to do some computations on the data being before writing them to memory. This is contrary to the copy mode which measures transfer rates without doing any additional arithmetic.

9.5 Evaluation of the complete framework

We deployed virtual honeypots in a low security production network of Universidad Politécnica de Madrid (UPM). Owing to the IP address resource limitation, we only applied two unused IP addresses registered in DNS for deploying virtual honeypots. Owing to the current security requirements of the campus network of UPM, we cannot deploy HIHs in the production network, and the LIHs, however, can only provide a little interaction to the attackers. Thus, we choose to deploy two virtual MIHs: Dionaea and Amun. After the data capture for 10 days, from December 17 to December 26, we calculated the statistic of data capture by the two MIHs in Table 6.

The statistic shows that the Dionaea suffered 6758 incidents and 21 of them hit the emulated vulnerabilities, whereas the Amun suffered 278 incidents and 15 of them hit the emulated vulnerabilities. The most popular vulnerable service is FTP using port 21. Thus, the production system should focus on protecting the FTP service from being attacked. Amun even captured one complete attack on port 21 and recorded the attacker's payloads:

```

2015-12-26 04:38:08,031 INFO [vuln_ftp] Attacker:
62.210.207.107 Message: ['USER anonymous\r\n'] Bytes: 16
Stage: FTPD_STAGE1
2015-12-26 04:38:08,066 INFO [vuln_ftp] Attacker:
62.210.207.107 Message: ['PASS anonymous@\r\n'] Bytes: 17
Stage: FTPD_STAGE1
2015-12-26 04:38:08,101 INFO [vuln_ftp] Attacker:
62.210.207.107 Message: ['CWD /\r\n'] Bytes: 7 Stage:
FTPD_STAGE2
2015-12-26 04:38:08,135 INFO [vuln_ftp] Attacker:
62.210.207.107 Message: ['TYPE A\r\n'] Bytes: 8 Stage:
FTPD_STAGE2.

```

We lookup this IP address and find the attacker is from Ile-de-France, Paris, France. Besides, we also lookup the busiest attacker using IP address 61.216.2.14, which were captured by both of the two honeypots, and it is from Taipei, Taiwan. Another busy attacker using IP address 83.34.220.39 locates in Pais Vasco, Gernika-Lumo, Spain. In addition, the Amun captured 30 unknown shellcodes which can be used for further investigation and improve the vulnerability emulation.

10 Conclusion and future work

A honeypot is deliberately used to be probed, attacked and compromised, in order to protect production systems, as well as investigate the well known and, especially, the unknown attacks. Owing to the requirement of applying heterogeneous honeypots to various attacks, in this paper, we propose a versatile virtual honeynet management tool called Honeyvers that can configure heterogeneous honeypots based on the generic language called TIHDL. Thanks to the combination of VNX and Honeyd, Honeyvers can emulate various individual honeypots. We also propose different honeynet deployment strategies for all the LIHs, MIHs and HIHs in terms of their design and application features. The experimental results show that the Honeyvers can be used to quickly and flexibly deploy different virtual honeynets. We apply the Honeyvers to deploy virtual honeypots in real production network as well, which validates the framework performs well in capturing real attacks. Admittedly, the performance of data capture mainly depends on the selected underlying honeypot software, while Honeyvers focus on the upper management. Thus, in the future, we will consider integrating much more individual honeypots into the framework in order to improve the power of data capture. Furthermore, we plan to design a flexible and adaptive honeynet system based on this tool for deploying specific honeypots against specific attacks. A more powerful sensor that can obtain the information of a complex network topology is under consideration as well.

11 Acknowledgments

This research is supported in part by the National Natural Science Foundation of China (nos. 61440057, 61272087, 61363019 and 61073008), the Beijing Natural Science Foundation (nos. 4082016 and 4122039), the Sci-Tech Interdisciplinary Innovation and Cooperation Team Program of the Chinese Academy of Sciences, the Specialized Research Fund for State Key Laboratories. It is also partially funded with support from the Spanish MICINN (project RECLAMO, Virtual and Collaborative Honeynets based on Trust Management and Autonomous Systems applied to Intrusion Management, with codes TIN2011-28287-C02-01 and TIN2011-28287-C02-02) and the European Commission (FEDER/ERDF).

12 References

- [1] 'The Value of Honeybots, Part One: Definitions and Values of Honeybots'. Available at <http://www.symantec.com/connect/articles/value-honeybots-part-one-definitions-and-values-honeybots>, accessed 16 June 2015
- [2] Provos, N.: 'A virtual honeypot framework'. SSYM'04 Proc. of the 13th Conf. on USENIX Security Symp., 2004, vol. 13
- [3] GobelJ.: 'Amun: automatic capturing of malicious software. ', Laboratory for Dependable Distributed Systems, University of Mannheim; , Germany: , 2010Technical Report
- [4] Kuwatly, L., Sraji, M., Al Masri, Z., *et al.*: 'A dynamic honeypot design for intrusion detection'. Proc. of the IEEE/ACS Int. Conf. on Pervasive Services, 2004, pp. 95–104
- [5] Hecker, C., Nance, K.L., Hay, B.: 'Dynamic honeypot construction'. Proc. of the Tenth Colloquium for Information Systems Security Education, 5–8 June 2006
- [6] Hecker, C., Hay, B.: 'Automated honeynet deployment for dynamic network environment'. Proc. of 46th Hawaii Int. Conf. on System Sciences (HICSS), 2013, pp. 4880–4889
- [7] Vrabie, M., Ma, J., Chen, J., *et al.*: 'Scalability, fidelity, and containment in the Potemkin virtual honeyfarm'. Proc. of the 20th ACM Symp. on Operating Systems Principles, 2005, pp. 148–162
- [8] Goldsack, P., Guijarro, J., Loughran, S., *et al.*: 'The SmartFrog configuration management framework', *SIGOPS Oper. Syst. Rev.*, 2009, **43**, (1), pp. 16–25
- [9] Capalik, A.: 'Next-generation honeynet technology with real-time forensics for U.S. Defense'. Proc. of IEEE Military Communications Conf. (MILCOM), October 2007, pp. 1–7
- [10] Pfoh, J., Schneider, C., Eckert, C.: 'Nitro: hardware-based system call tracing for virtual machines'. Proc. of the Sixth Int. Conf. on Advances in Information and Computer Security (IWSEC'11), Berlin, Heidelberg, 2011, pp. 96–112
- [11] Memari, N., Hashim, S.J.B., Samsudin, K.B.: 'Towards virtual honeynet based on LXC virtualization'. Proc. of IEEE Region 10 Symp., April 2014, pp. 496–501
- [12] Fan, W., Fernández, D., Villagrà, V.A.: 'Technology independent honeynet description language'. Proc. of Third Int. Conf. on Model-Driven Engineering and Software Development, MODELSWARD, Angers, France, 9–11 February 2015
- [13] Fernandez, D., Cordero, A., Somavilla, J., *et al.*: 'Distributed virtual scenarios over multi-host Linux environments'. Proc. of Fifth Int. DMTF Academic Alliance Workshop on Systems and Virtualization Management (SVM), October 2011, pp. 1–8